Lecture 3

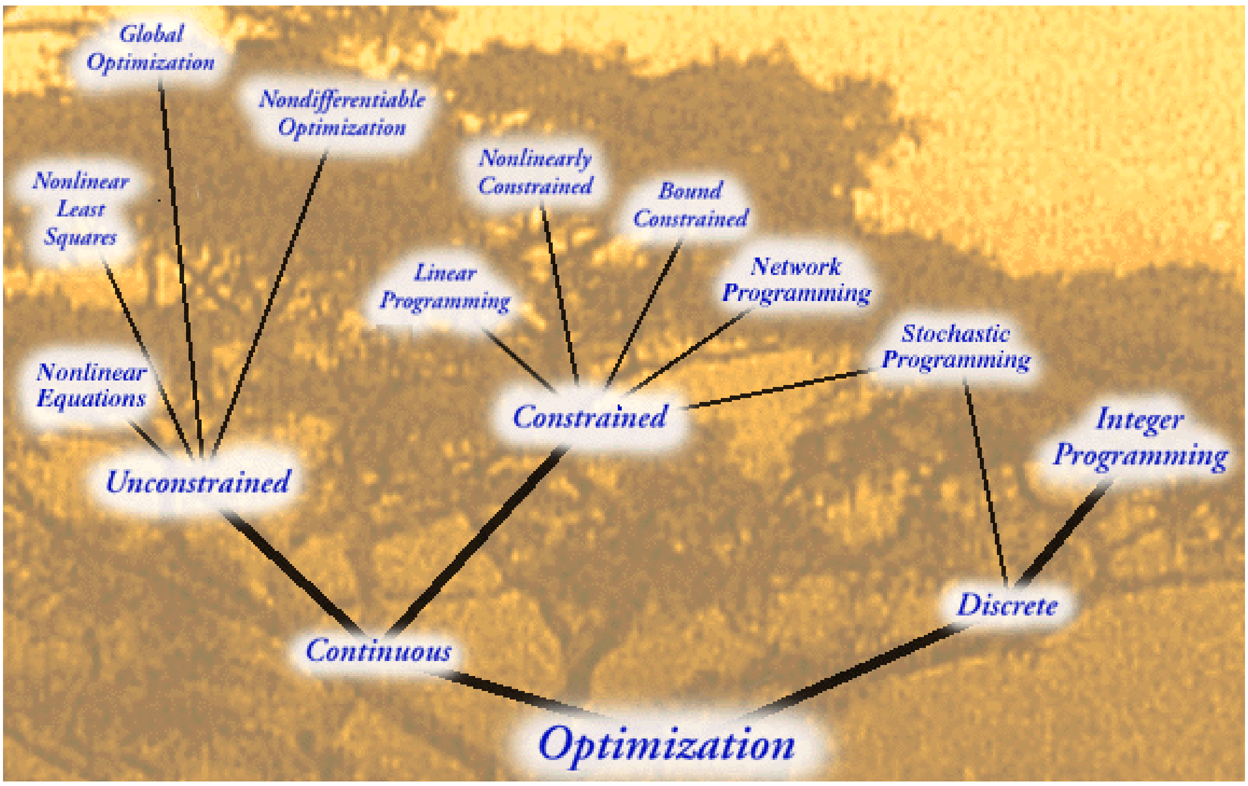
C7B Optimization Hilary 2011 A. Zisserman

Cost functions with special structure:

- Levenberg-Marquardt algorithm
- Dynamic Programming
  - chains
  - applications

First: review Gauss-Newton approximation

The Optimization Tree



Summary of minimizations methods

Update $\mathbf{x}_{n+1} = \mathbf{x}_n + \boldsymbol{\delta}\mathbf{x}$

1. Newton.

$$\mathtt{H}\,\boldsymbol{\delta}\mathbf{x} = -\mathbf{g}$$

2. Gauss-Newton.

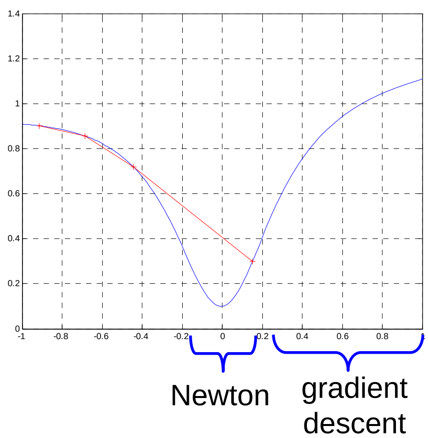
$$2\mathtt{J}^\top\mathtt{J}\,\boldsymbol{\delta}\mathbf{x} = -\mathbf{g}$$

3. Gradient descent.

$$\lambda\boldsymbol{\delta}\mathbf{x} = -\mathbf{g}$$

Levenberg-Marquardt algorithm

- Away from the minimum, in regions of negative curvature, the Gauss-Newton approximation is not very good.
- In such regions, a simple steepest-descent step is probably the best plan.
- The Levenberg-Marquardt method is a mechanism for varying between steepest-descent and Gauss-Newton steps depending on how good the $\mathtt{J}^\top\mathtt{J}$ approximation is locally.



- The method uses the modified Hessian

$$\mathtt{H}\,(\mathbf{x}, \lambda) = 2\mathtt{J}^{\top}\mathtt{J} + \lambda\mathtt{I}$$

- When λ is small, $\mathtt{H}$ approximates the Gauss-Newton Hessian.

- When λ is large, $\mathtt{H}$ is close to the identity, causing steepest-descent steps to be taken.

LM Algorithm

$$\mathtt{H}\,(\mathbf{x}, \lambda) = 2\mathtt{J}^{\top}\mathtt{J} + \lambda\mathtt{I}$$

1. Set $\lambda = 0.001$ (say)

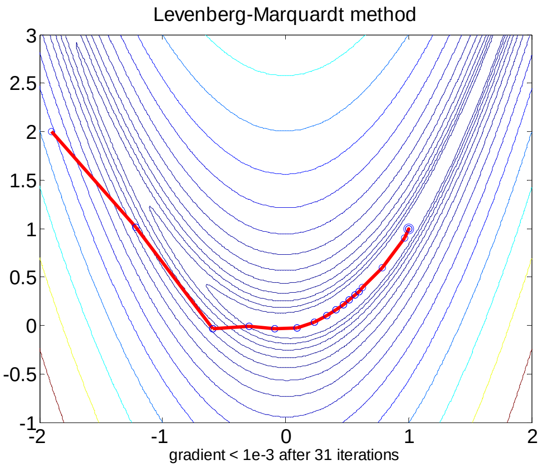
2. Solve $\boldsymbol{\delta}\mathbf{x} = -H(\mathbf{x}, \lambda)^{-1}\,\mathbf{g}$

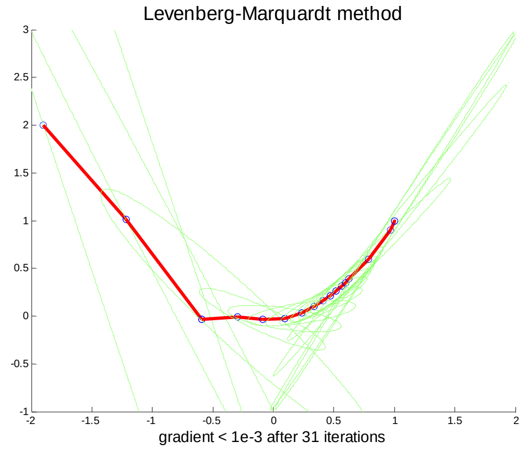
3. If $f(\mathbf{x}_n + \boldsymbol{\delta}\mathbf{x}) > f(\mathbf{x}_n)$, increase λ ($\times 10$ say) and go to 2.

4. Otherwise, decrease λ ($\times 0.1$ say), let $\mathbf{x}_{n+1} = \mathbf{x}_n + \boldsymbol{\delta}\mathbf{x}$, and go to 2.

Note : This algorithm does not require explicit line searches.

Example



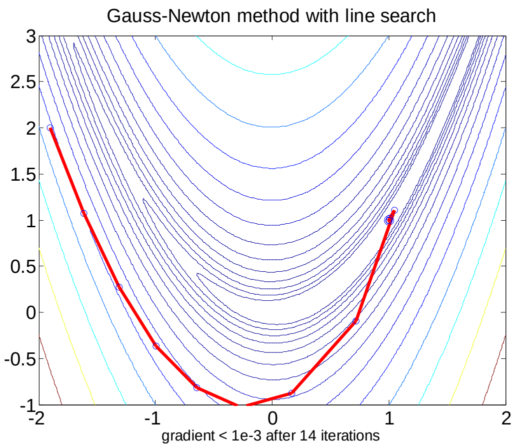


- Minimization using Levenberg-Marquardt (no line search) takes 31 iterations.

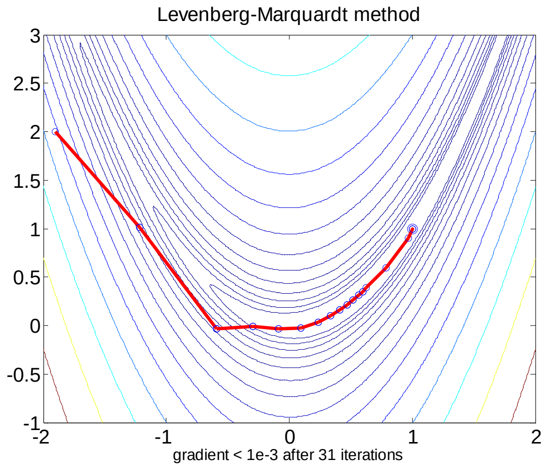
Matlab: lsqnonlin

Comparison

Gauss-Newton



Levenberg-Marquardt



- more iterations than Gauss-Newton, but
- no line search required,
- and more frequently converges

Case study – Bundle Adjustment (non-examinable)

Problem statement

- Given: n matching image points $\mathbf{x}^i_j$ over m views
- Find: the cameras P^i and the 3D points $\mathbf{X}_j$ such that $\mathbf{x}^i_j = P^i \mathbf{X}_j$

$$\min_{P^i\,\mathbf{X}_j} \sum_{j \in \text{points}} \sum_{i \in \text{views}} d\left(\mathbf{x}^i_j,\ P^i \mathbf{X}_j\right)^2$$

Notation:

- A 3D point $\mathbf{X}_j$ is imaged in the "i" th view as

$$\mathbf{x}^i_j = P^i \mathbf{X}_j$$

P : 3×4 matrix

$\mathbf{X}$: 4-vector

$\mathbf{x}$: 3-vector

Number of parameters

$$\min_{P^i\,\mathbf{X}_j} \sum_{j \in \text{points}} \sum_{i \in \text{views}} d\left(\mathbf{x}^i_j,\ P^i \mathbf{X}_j\right)^2$$

- for each camera there are 6 parameters
- for each 3D point there are 3 parameters

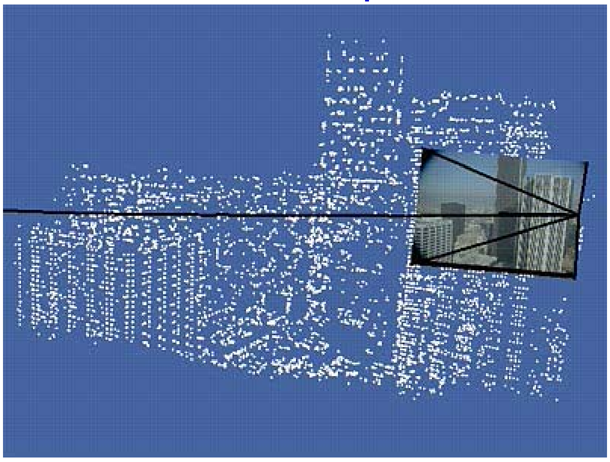
a total of 6 m + 3 n parameters must be estimated

- e.g. 50 frames, 1000 points: 3300 unknowns

Example

image sequence

cameras and points



Sparse form of the Jacobian matrix

- Image point $\mathbf{x}_j^i$ does not depend on the parameters of any camera other than $\mathtt{P}^i$.
- Thus,

$$\partial \mathbf{x}_j^i / \partial \mathtt{P}^k = 0$$

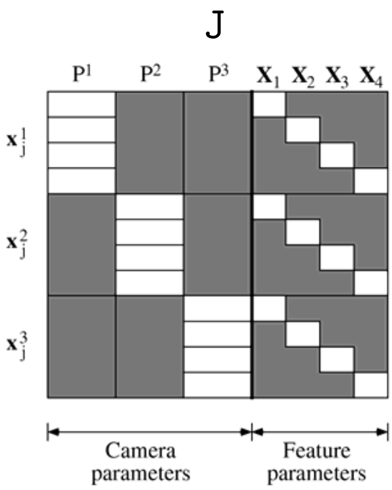
unless $i = k$.

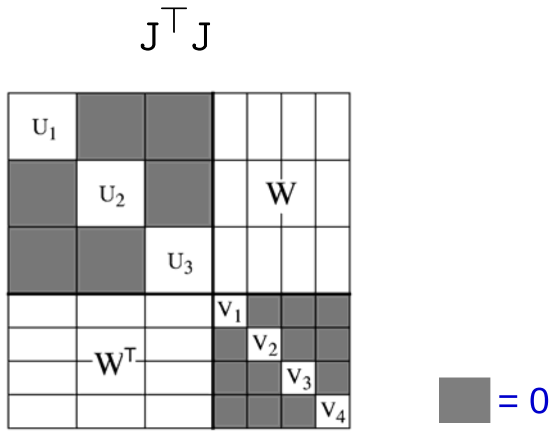
- Similarly, image point $\mathbf{x}_j^i$ does not depend on any 3D point except $\mathbf{X}_j$.

$$\partial \mathbf{x}_j^i / \partial \mathbf{X}_k = 0$$

unless $j = k$.

Form of the Jacobian and Gauss-Newton Hessian for the bundle-adjustment problem consisting of 3 cameras and 4 points.





By taking advantage of this sparse form, one iterative update of the LM algorithm

- $\mathtt{H}(\mathbf{x},\lambda) = 2\mathtt{J}^\top\mathtt{J} + \lambda\mathtt{I}$
- Solve $\boldsymbol{\delta}\mathbf{x} = -H(\mathbf{x},\lambda)^{-1}\,\mathbf{g}$

can be solved in $O(N)$ rather than $O(N^3)$, where N is the total number of parameters

Application: Augmented reality

original sequence

Augmentation

Dynamic programming

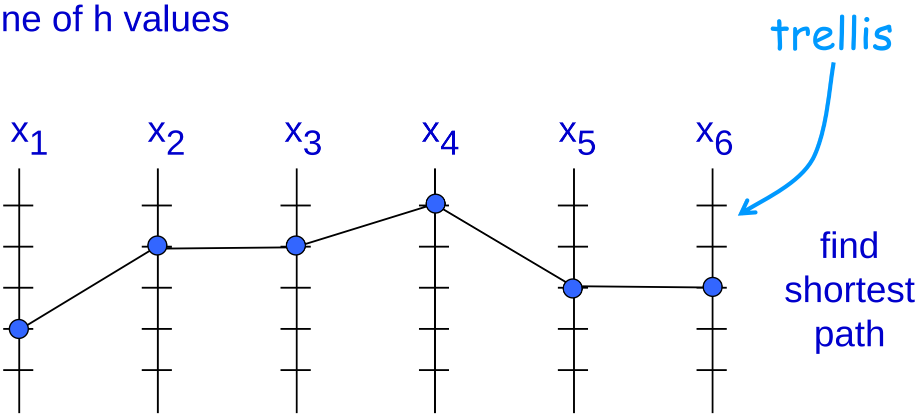
- Discrete optimization
- Each variable x has a finite number of possible states
- Applies to problems that can be decomposed into a sequence of stages
- Each stage expressed in terms of results of fixed number of previous stages
- The cost function need not be convex
- The name “dynamic” is historical
- Also called the “Viterbi” algorithm

Consider a cost function $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi_i(x_{i-1}, x_i)$$

where x_i can take one of h values

e.g. h=5, n=6



$$f(\mathbf{x}) = \left\{ \begin{array}{c} m_1(x_1) + m_2(x_2) + m_3(x_3) + m_4(x_4) + m_5(x_5) + m_6(x_6) \\ \phi(x_1, x_2) + \phi(x_2, x_3) + \phi(x_3, x_4) + \phi(x_4, x_5) + \phi(x_5, x_6) \end{array} \right.$$

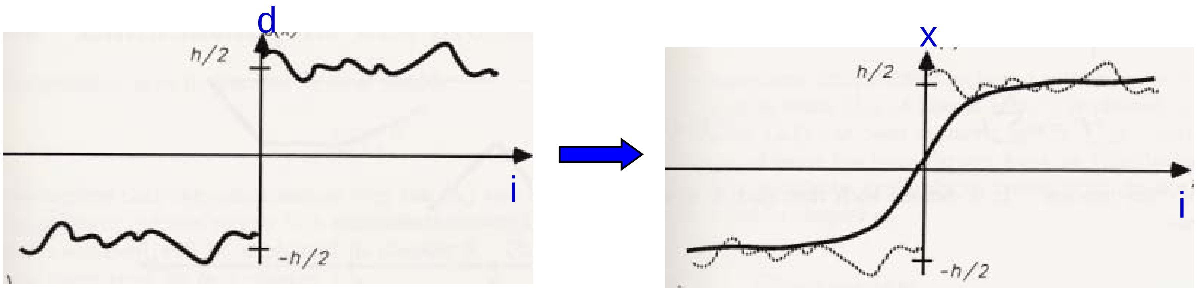
Complexity of minimization:

- exhaustive search $O(h^n)$
- dynamic programming $O(nh^2)$

Example 1

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i)$$

$$f(\mathbf{x}) = \sum_{i=1}^{n} \underbrace{(x_i - d_i)^2}_{\text{closeness to measurements}} + \sum_{i=2}^{n} \underbrace{\lambda^2 (x_i - x_{i-1})^2}_{\text{smoothness}}$$



Motivation: complexity of stereo correspondence

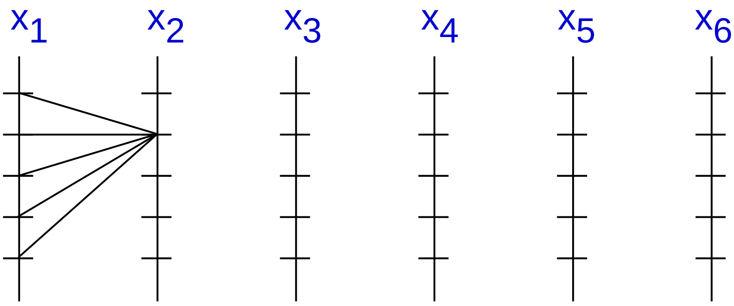
Objective: compute horizontal displacement for matches between left and right images

x_i is spatial shift of i'th pixel $\rightarrow h = 40$

$\mathbf{x}$ is all pixels in row $\rightarrow n = 256$

Complexity $O(40^{256})$ vs $O(256 \times 40^2)$

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i)$$



Key idea: the optimization can be broken down into n sub-optimizations

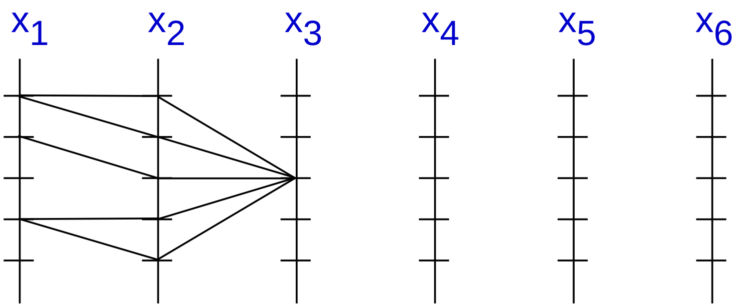
Step 1: For each value of x_2 determine the best value of x_1

- Compute

$$\begin{aligned} S_2(x_2) &= \min_{x_1}\{m_2(x_2) + m_1(x_1) + \phi(x_1, x_2)\} \\ &= m_2(x_2) + \min_{x_1}\{m_1(x_1) + \phi(x_1, x_2)\} \end{aligned}$$

- Record the value of x_1 for which $S_2(x_2)$ is a minimum

To compute this minimum for all x_2 involves $O(h^2)$ operations



Step 2: For each value of x_3 determine the best value of x_2 **and** x_1

- Compute

$$S_3(x_3) = m_3(x_3) + \min_{x_2}\{S_2(x_2) + \phi(x_2, x_3)\}$$

- Record the value of x_2 for which $S_3(x_3)$ is a minimum

Again, to compute this minimum for all x_3 involves $O(h^2)$ operations
Note $S_k(x_k)$ encodes the lowest cost partial sum for all nodes up to k which have the value x_k at node k, i.e.

$$S_k(x_k) = \min_{x_1, x_2, \ldots, x_{k-1}} \sum_{i=1}^{k} m_i(x_i) + \sum_{i=2}^{k} \phi(x_{i-1}, x_i)$$

Viterbi Algorithm

- Initialize $S_1(x_1) = m_1(x_1)$
- For $k = 2 : n$

$$\begin{aligned} S_k(x_k) &= m_k(x_k) + \min_{x_{k-1}}\{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\} \\ b_k(x_k) &= \arg\min_{x_{k-1}}\{S_{k-1}(x_{k-1}) + \phi(x_{k-1}, x_k)\} \end{aligned}$$

- Terminate

$$x_n^* = \arg\min_{x_n} S_n(x_n)$$

- Backtrack

$$x_{i-1} = b_i(x_i)$$

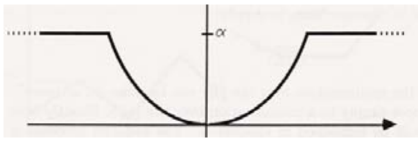
Complexity $O(nh^2)$

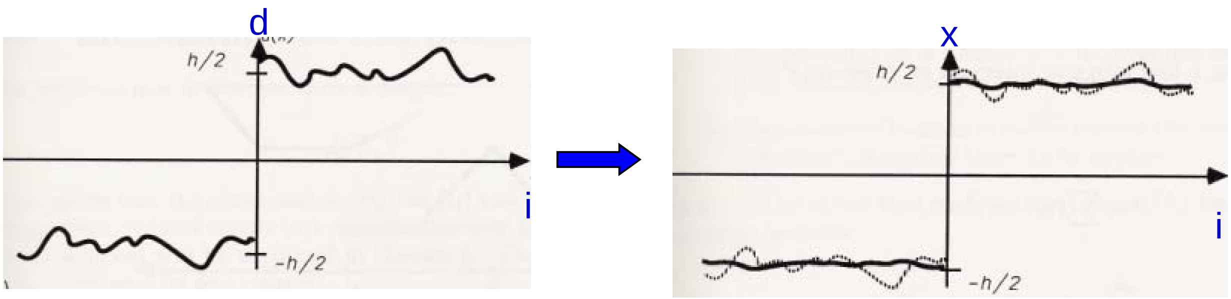
Example 2

$$f(\mathbf{x}) = \sum_{i=1}^{n}(x_i - d_i)^2 + \sum_{i=2}^{n} g_{\alpha,\lambda}(x_i - x_{i-1})$$

where

$$g_{\alpha,\lambda}(\Delta) = \min(\lambda^2\Delta^2, \alpha) = \begin{cases} \lambda^2\Delta^2 & \text{if } |\Delta| < \sqrt{\alpha}/\lambda \\ \alpha & \text{otherwise.} \end{cases}$$





Note, f(x) is not convex

Note

This type of cost function often arises in MAP estimation

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}|\underbrace{\mathbf{y}}_{\text{measurements}})$$

measurements

$$= \arg\max_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$$

Bayes' rule

$$\sim \prod_i^n e^{-\frac{(x_i-y_i)^2}{2\sigma^2}} e^{-\beta^2(x_i-x_{i-1})^2}$$

e.g. for Gaussian measurement errors, and first order smoothness

Use negative log to obtain a cost function of the form

$$f(\mathbf{x}) = \sum_{i=1}^n \underbrace{(x_i - y_i)^2}_{\text{from likelihood}} + \sum_{i=2}^n \underbrace{\lambda^2(x_i - x_{i-1})^2}_{\text{from prior}}$$

Where can DP be applied?

Dynamic programming can be applied when there is a linear ordering on the cost function (so that partial minimizations can be computed).

Example Applications:

1. Text processing: String edit distance
2. Speech recognition: Dynamic time warping
3. Computer vision: Stereo correspondence
4. Image manipulation: Image re-targeting
5. Bioinformatics: Gene alignment

Application I: string edit distance

The edit distance of two strings, s1 and s2, is the minimum number of single character mutations required to change s1 into s2, where a mutation is one of:

1. substitute a letter	(kat → cat)	cost = 1
2. insert a letter	(ct → cat)	cost = 1
3. delete a letter	(caat → cat)	cost = 1

Example: d(opimizateon, optimization)

```
op imizateon
|| |||||||||
optimization
||||||||||||
ccicccccccscc
```

'c' = copy, cost = 0

d(s1,s2) = 2

Complexity

- for two strings of length m and n, exhaustive search has complexity O(3^{m+n})

- dynamic programming reduces this to O(mn)

Using string edit distance for spelling correction

1. Check if word w is in the dictionary D
2. If it is not, then find the word x in D that minimizes d(w, x)
3. Suggest x as the corrected spelling for w

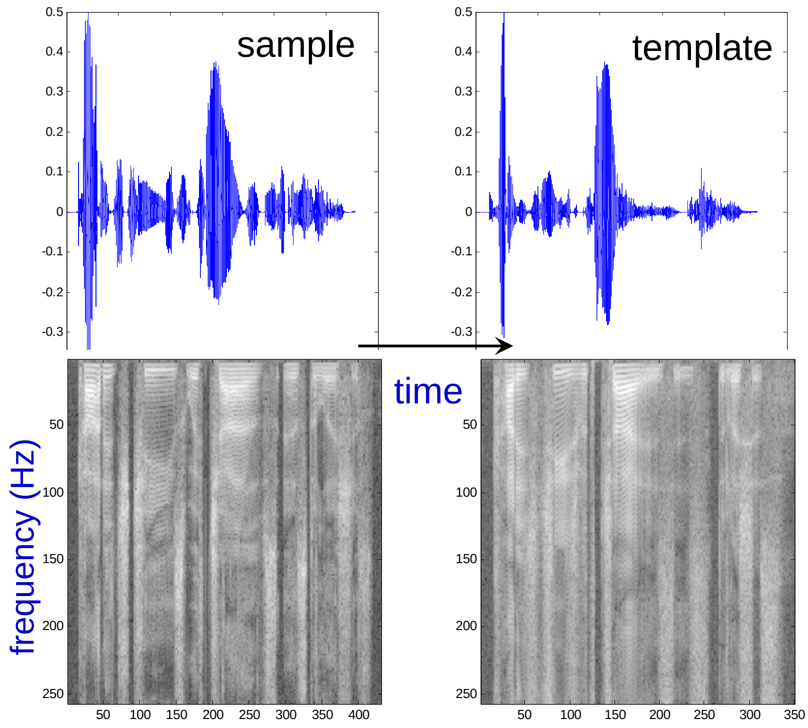
Note: step 2 appears to require computing the edit distance to all words in D, but this is not required at run time because edit distance is a metric, and this allows efficient search.

Mispelling	Word	ED	Detail
committment	commitment	1	committment $\rightarrow$ commitment
tommorrow	tomorrow	1	tommorrow $\rightarrow$ tomorrow
saftey	safety	2	saftey $\rightarrow$ safty $\rightarrow$ safety

Application II: Dynamic Time Warp (DTW)

Objective: temporal alignment of a sample and template speech pattern

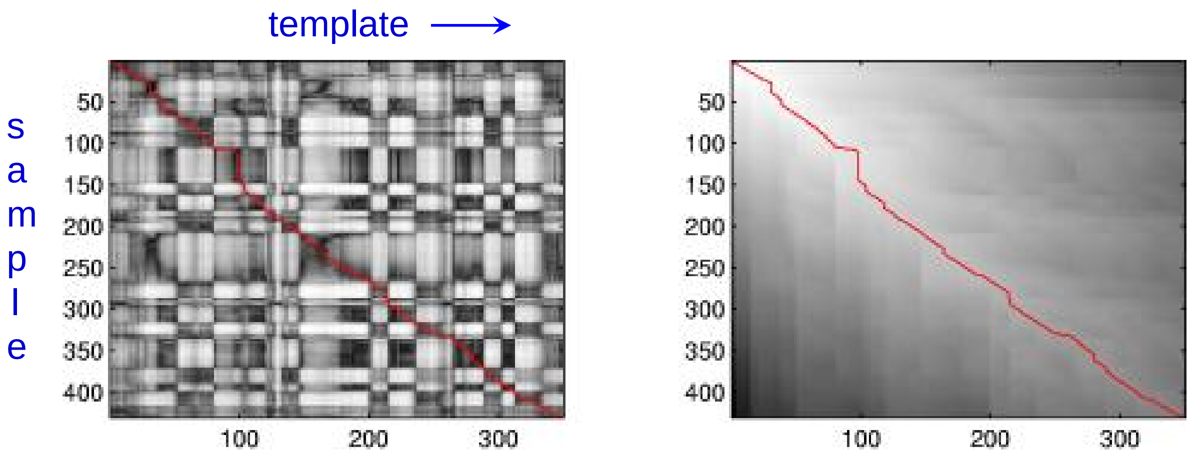
audio



log(STFT)

short term
Fourier
transform

warp to match
`columns' of
log(STFT) matrix



x_i is time shift of *i* th column

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i)$$

quality of match

cost of allowed moves

→ (1, 0)

↓ (0, 1)

↘ (1, 1)

Application III: stereo correspondence

Objective: compute horizontal displacement for matches between left and right images

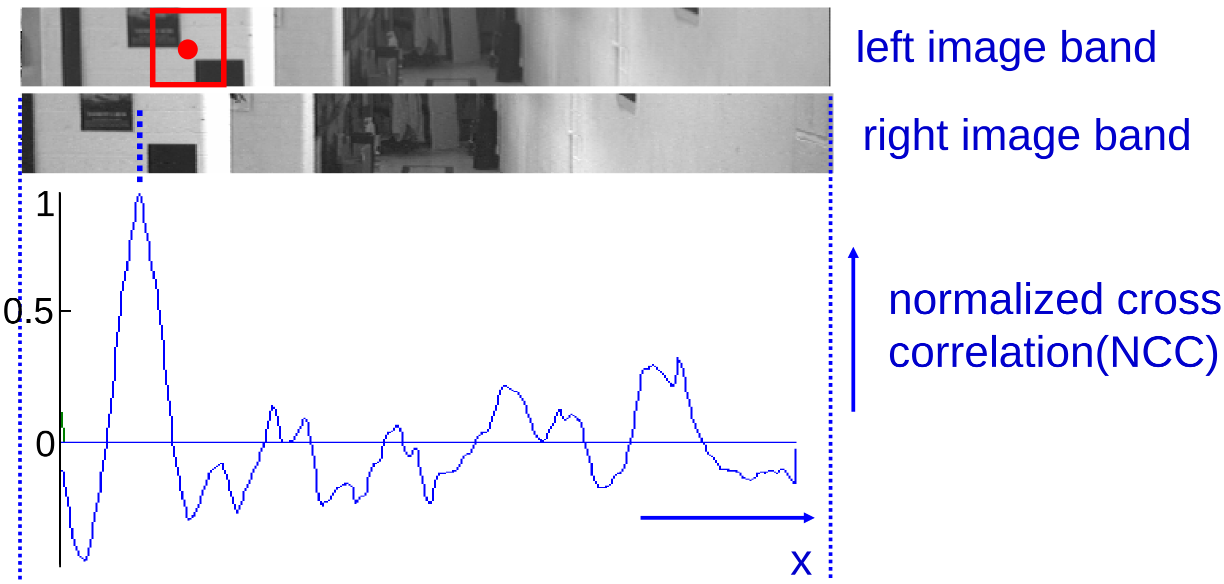
x_i is spatial shift of *i* th pixel

$$f(\mathbf{x}) = \sum_{i=1}^{n} m_i(x_i) + \sum_{i=2}^{n} \phi(x_{i-1}, x_i)$$

quality of match

uniqueness, smoothness

$$m(x) = \alpha(1 - \text{NCC})^2$$

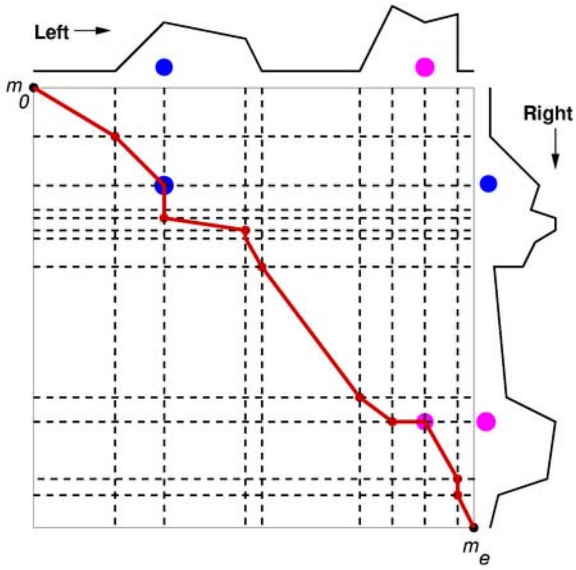


NCC of square image regions at offset (disparity) x

- Arrange the raster intensities on two sides of a grid

- Crossed dashed lines represent potential correspondences

- Curve shows DP solution for shortest path (with cost computed from f(x))

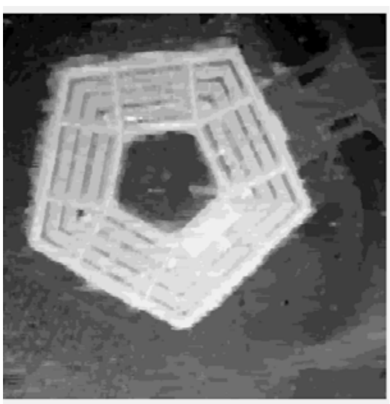


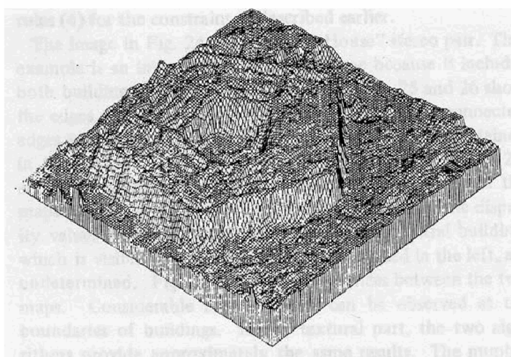
Pentagon example

left image

right image

range map





Real-time application – Background substitution

Left view

Right view

Input

Results

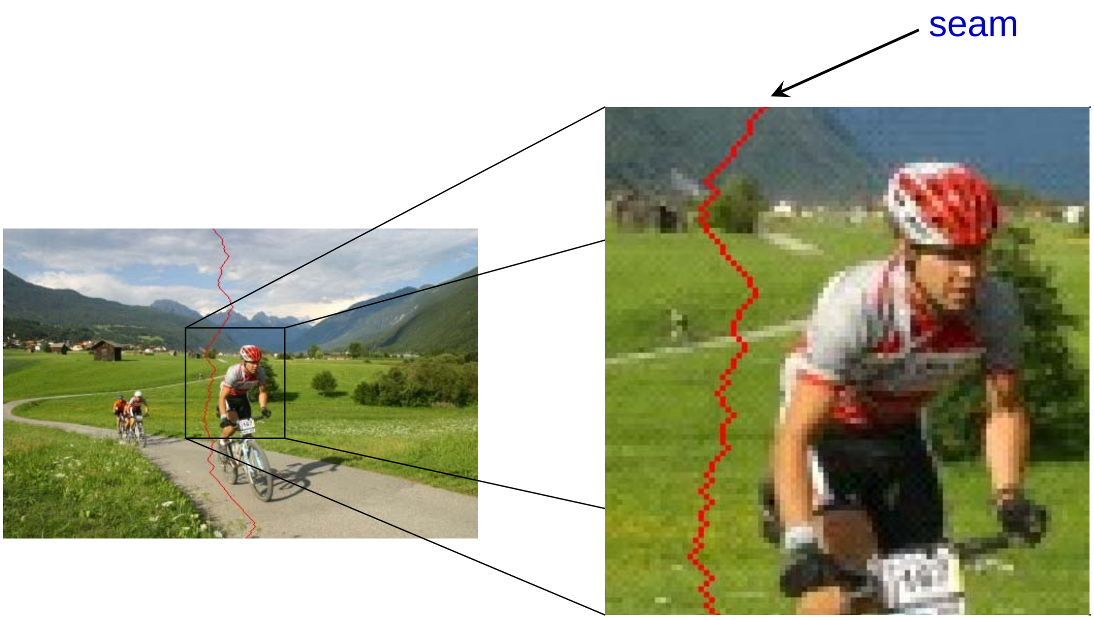
input left view

Background substitution 1

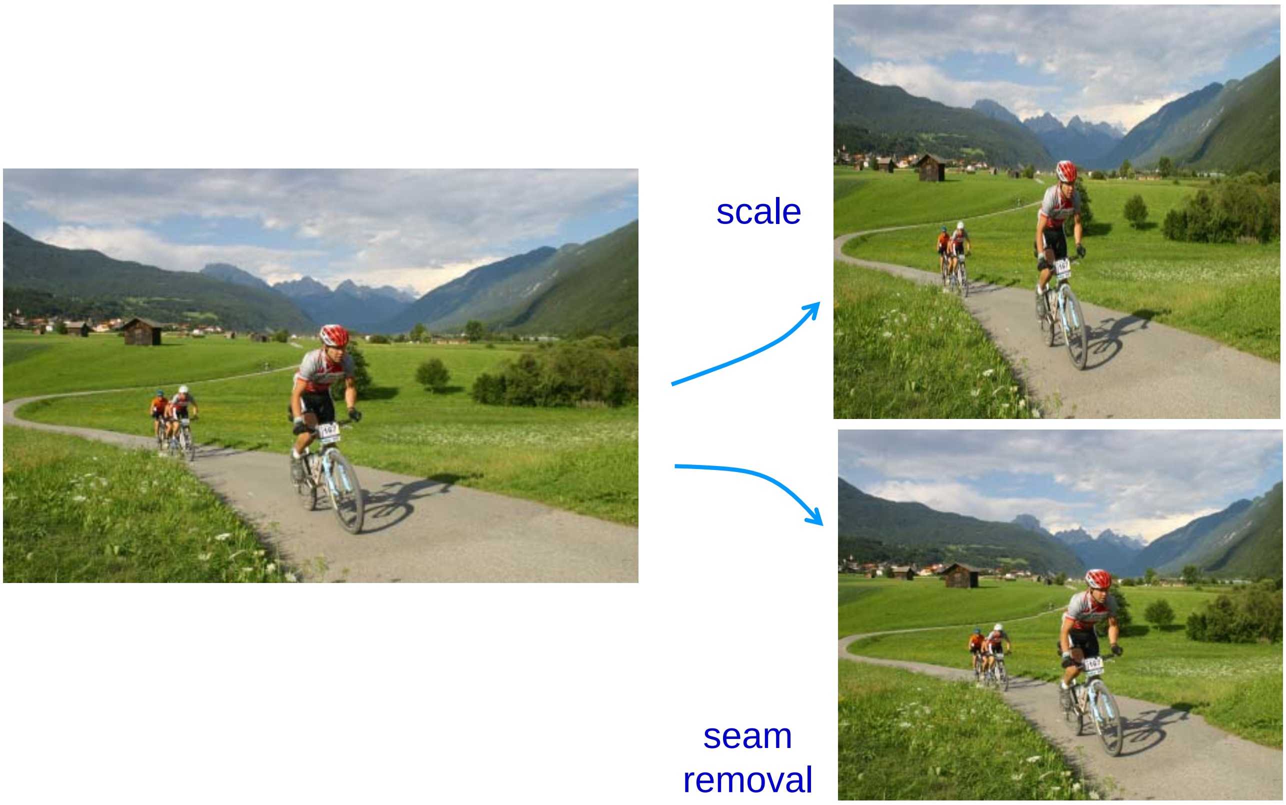
Background substitution 2

Application IV: image re-targeting

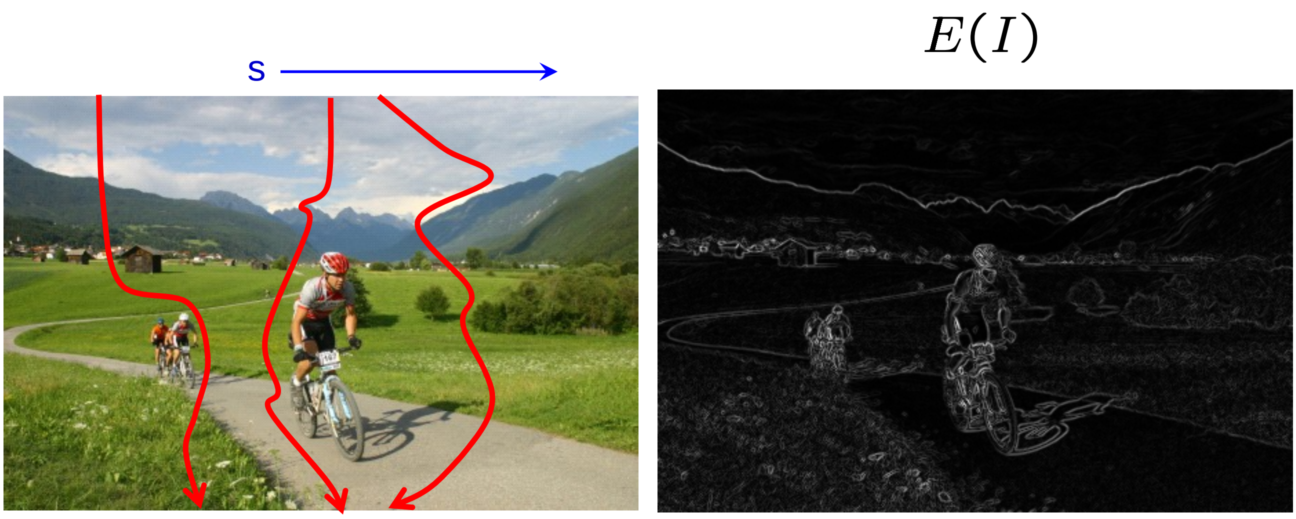
- Remove image “seams” for imperceptible aspect ratio change



Seam Carving for Content-Aware Image Retargeting. *Avidan and Shamir, SIGGRAPH, San-Diego, 2007*



Finding the optimal seam – s



$$E(I) = |\partial I / \partial x| + |\partial I / \partial y| \rightarrow s^* = \arg\min_s E(s)$$